FakeInf: Selective Deep Neural Network Inference for Latency and Energy-Aware Model Serving Pipelines

Demetris Trihinas
Computer Science
University of Nicosia
Nicosia, Cyprus
trihinas.d@unic.ac.cy

Moysis Symeonides
Computer Science
University of Cyprus
Nicosia, Cyprus
msymeo03@ucy.ac.cy

Nicolae Cleju
Technical University of Iasi
Iasi, Romania
ncleju@etti.tuiasi.ro

George Pallis
Computer Science
University of Cyprus
Nicosia, Cyprus
pallis@ucy.ac.cy

Marios Dikaiakos
Computer Science
University of Cyprus
Nicosia, Cyprus
mdd@ucy.ac.cy

Abstract

Recent advances in 5G networks and edge computing are enabling low-latency and AI-powered services in close proximity to end users. However, the growing complexity of Deep Learning (DL) models is threatening the vision of EdgeAI, where real-time inference demands substantial computational power, excessive usage of energy, and imposes heavy model-update traffic that overwhelm resource-constrained multi-access edge computing (MECs) nodes. In this paper, we present FakeInf, a framework that supports EdgeAI applications delivering DL-based video stream inference. FakeInf adds a lightweight decision module to DL model-serving pipelines that tracks data volatility and, using probabilistic reasoning, decides for streamed input whether to run the full model or "fake it" by relying on low-cost statistical estimations. This selective execution reduces network traffic, latency, and energy usage while maintaining Quality-of-Service (QoS) within user-defined limits. To demonstrate the efficacy of FakeInf, we integrate it with a real-world smart traffic system hosted on a MEC. FakeInf reduces application latency by 59%, network traffic by 71%, computational overhead by 66%, and energy by 72% while incurring only a modest reduction of 4–6% in the accuracy of the analytic insights emitted. FakeInf also allowed the pipeline to process 2x more video streams compared to the baseline without creating inference bottlenecks.

CCS Concepts

• **Computer systems organization** → **Cloud computing**; • **Networks** → *In-network processing*; • **Computing methodologies** → *Machine learning*.

Keywords

Edge Computing, Deep Learning, B5G networks

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.
UCC '25, Nantes, France
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2285-1/25/12
https://doi.org/10.1145/3773274.3774270

ACM Reference Format:
Demetris Trihinas, Moysis Symeonides, Nicolae Cleju, George Pallis, and Marios Dikaiakos. 2025. FakeInf: Selective Deep Neural Network Inference for Latency and Energy-Aware Model Serving Pipelines. In *2025 IEEE/ACM 18th International Conference on Utility and Cloud Computing (UCC '25), December 01–04, 2025, Nantes, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3773274.3774270

1 Introduction

The advent of 6G networks promises to build on the ultra-low-latency, high-throughput, and pervasive connectivity of 5G by embedding AI capabilities as native network functions [11]. Deploying AI, such as Deep Learning (DL) models, on multi-access edge computing nodes (MECs) is advancing the vision of edge intelligence, or simply EdgeAI. Through this paradigm, users gain in situ and impromptu access to network-hosted AI services, while simultaneously offloading data from backhaul links that connect MEC nodes to central cloud resources [28]. The intersection of these technological advances is already addressing pressing socioeconomical challenges, including resilience and sustainability, as the network traffic attributed to EdgeAI is expected to scale 3x by 2029 [1].

Examples of new and emerging use-cases driving EdgeAI include intelligent traffic control [5], immersive navigation with augmented reality [8], and digital twin assisted surgical procedures [2]. However, the demand for such sophisticated AI services is forcing DL models to expand rapidly in parameter count, as well as layer depth and width [18]. This increasing complexity, exemplified by billion-parameter transformers and multimodal networks, overwhelms the limited compute, memory, and energy resources of MEC nodes [23][31]. More surprisingly, the energy consumption of state-of-the-art DL models is doubling at an annual rate [7][13]. As a result, the vision towards delivering low-latency and energy-efficient inference at the network edge is coming under strain.

These risks are particularly evident in smart city applications that require continuous inference on live data streams, such as object detection running on multiple video feeds from geo-distributed IP cameras for crowd monitoring and vehicle traffic management. This directly impacts the real-time processing of high-volume video streams on computationally constrained MEC infrastructures [20]. In such cases, the underlying 5G/B5G network may be able to handle

HD video streams due to its high-bandwidth connections, but the DL inference process itself often becomes a bottleneck, limiting the scalability and sustainability of MEC deployments [29].

To address this challenge, different techniques have been proposed. A traditional approach is the horizontal scaling of MECs. However, MEC scaling is not as simple as elastically adjusting virtual offerings in the cloud, facing challenges related to adding physical infrastructure in the mobile network [9]. Focusing on more cost-effective solutions at the DL model level, others suggest applying pruning or quantization (e.g., reduce number and precision of model parameters) [10]. Unfortunately, these come with a fixed cost that can rapidly reduce accuracy during inference, even in cases where MEC nodes face no duress. To overcome this limitation, a more versatile technique is model-swapping, in which a repository of pretrained model variants is available and based on the computational capacity of the edge node, a variant is selected to ensure the latency requirements set by the service operators are maintained [24]. Still, a key challenge is that the number of model variants preloaded in (GPU) memory is limited in practice.

To ensure latency and energy requirements during task inference for DL model-serving pipelines hosted on MEC nodes, we introduce FakeInf[1]. Specifically, the FakeInf framework instantiates a decision-making module within the DL model-serving pipeline that embeds a lightweight algorithmic mechanism designed to autonomously adapt the temporal intensity of the runtime inference process. To do so, FakeInf uses low-cost statistical estimators that adopt probabilistic reasoning to monitor the runtime volatility of the analytical insights produced by the DL model (e.g., object detection counts) and determine how often DL inference should be applied to the video stream. When video stream analytics exhibit low volatility phases, the estimator identifies this condition and the FakeInf algorithmic mechanism recommends a time frame of X video frames during which inference is not triggered, using instead the estimator's output. By introducing the FakeInf decision-making module, the volume of data emitted over the mobile network is significantly reduced. This also provides valuable "breathing space" to the computational units (e.g., GPU/CPU) of the underlying MEC node and reduces the node's energy footprint.

To show the efficacy and efficiency of the FakeInf framework, we conducted a series of field trials integrating FakeInf with a real-world smart traffic service hosted on a B5G MEC node provided by Orange, Romania. The smart traffic service entails monitoring a road intersection in Iasi with IP cameras. The video feed is streamed to a DL model-serving pipeline for object detection using a vision model and subsequent post-processing to emit, among other analytic insights, vehicle and pedestrian counts. The results show that FakeInf reduces application latency by 59%, network traffic by 71%, GPU usage by 66%, and energy by 72% with only a modest reduction of 4-6% in accuracy of the analytics emitted. FakeInf also allowed the pipeline to process 2x more video streams compared to the baseline before the MEC node becomes a computational bottleneck and cannot serve in-time and accurate analytic insights.

The rest of this paper is structured as follows: Section 2 provides a background and motivation overview. Section 3 describes the algorithmic mechanisms behind FakeInf, while Section 4 introduces implementation and integration aspects. Section 5 provides a comprehensive evaluation study. Section 6 an overview of related work, while Section 7 concludes the paper and outlines future work.

[1] https://github.com/unic-ailab/FakeInf

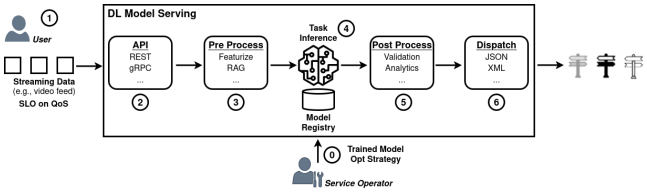

Figure 1: Abstract View of a DL Model-Serving Pipeline

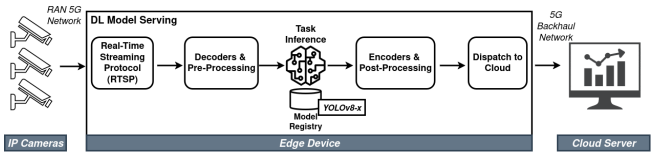

Figure 2: Example of EdgeAI Smart Traffic Analytics Service

2 Background and Motivation

2.1 Model Serving

DL model-serving refers to the offering of a software stack enabling developers to deploy binary artifacts of trained neural networks in production-level environments. Models are exposed through APIs for users to submit inference requests that deliver prediction or classification responses based on given input data [3]. Popular frameworks such as TensorFlow-Serving[2], and Nvidia DeepStream[3] ease model lifecycle management (e.g., versioning) and notably, during inference handle request scheduling so that downstream applications can consume model outputs without embedding model logic. Moreover, tools such as Clipper [6] and Nexus [19] introduce multi-client request serving and support the definition of app-level SLOs for inference execution.

The workflow of a typical DL model-serving platform is as follows (Fig. 1): Initially, Service Operators register a (new) model version to the platform's repository ⓪. Clients then submit inference requests via public APIs (e.g., REST, gRPC) ①, supplying the input data and, if desired, specifying SLOs, such as an end-to-end latency below 20ms ②. Once the platform receives a request, it initiates pre-processing ③, which can include feature extraction, retrieval-augmented generation (RAG) to gather supplementary external data, and other transformations such as resolution downscaling. Next, the feature vector of the encoded and preprocessed input is fed to the DNN model to emit the inferred output ④. Subsequently, the system performs post-processing ⑤ to update analytic metrics (for example, object counts) and carry out any necessary validation. Finally, a dispatcher delivers the results for downstream consumption according to the user's requirements ⑥.

2.2 Edge-AI Traffic Management Use-Case

Let us highlight an EdgeAI application that motivates our work: a smart traffic-management system deployed on a B5G MEC node operated by Orange, Romania. In this setup, high-definition IP

[2] https://www.tensorflow.org/tfx/guide/serving
[3] https://developer.nvidia.com/deepstream-sdk

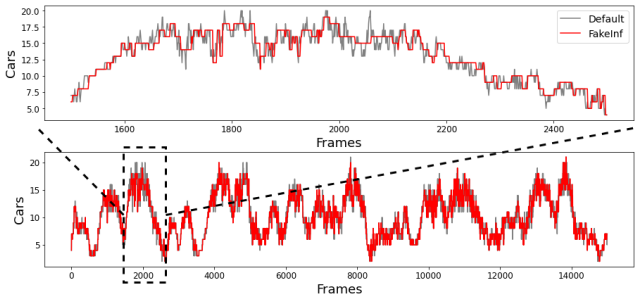

Figure 3: Example Car-Count Metric for Respective Use Case

cameras are mounted at a busy road intersection in Iasi, with these streaming video to a Nvidia DeepStream model-serving pipeline that performs real-time object detection using the largest YOLO-v8 model variant. The pipeline continuously updates vehicle and pedestrian counts that traffic operators visualize on a cloud-hosted dashboard to monitor traffic congestion and receive incident alerts. Figure 2 provides a high-level overview of the use-case pipeline. More details about the use-case implementation and operation are introduced in Section 4.

Figure 3 presents an exemplary visualization of the car-count metric. Specifically, it shows a 10-minute timeline of a video feed, including a 40-second zoomed-in segment at a random time interval (top plot). The frame rate is set to 25 frames per second, with the gray line highlighting the car-count values captured by the default execution of the EdgeAI service. At this fps, the system achieves its highest accuracy using YOLOv8-X (~90% precision). However, achieving this level of performance with YOLOv8-X (68.2M params) significantly stresses the GPU of the MEC node, increasing utilization (257B flops) and energy consumption (61 W/s) of the edge node. Moreover, examining the car-count trend in Fig. 3, raises an important consideration: *if the count varies only slightly over time, the benefit of continuously running the video feed through the DNN, diminishes*. This observation raises an important question: *can we maintain acceptable accuracy while reducing unnecessary computation by selectively skipping or approximating inferences?* Addressing this question leads us to propose *faking inference*, a technique that adaptively controls the inference execution based on the emitted analytic insights and the stream variability.

2.3 The "Fake Inference" Approach

DL model-serving must balance competing objectives. For example, users expect low-latency and high-accuracy, whereas Service Operators want to reduce operational expenditure. Energy consumption is a principal contributor to these costs directly associated with the computational power (GPU/CPU) employed during inference [16]. Approximation methods, such as model pruning or fixed frame sampling, can reduce energy usage and shorten inference latency, thereby benefiting both service operators and users. However, applying a fixed approximation often degrades accuracy to an unacceptable extent, even in cases where perhaps no latency reduction is required or requested. Consequently, the platform must adaptively decide *when* and for *how long* the approximation is employed.

This requirement is well suited to workloads that monitor physical processes, such as traffic at road intersections via IP cameras,

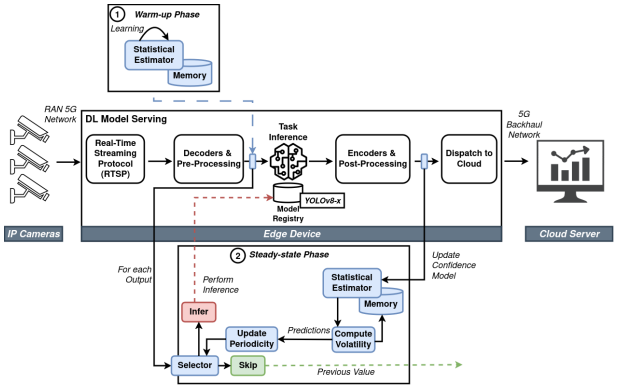

Figure 4: High-level Overview of FakeInf Framework

where successive video frames exhibit strong temporal similarity, as presented in Figure 3. Our "fake inference" approach capitalizes on this observation. Specifically, when the analytic insights extracted from a video stream display phases of low volatility, forwarding frames to the DNN for inference can be skipped. Conversely, when the stream's analytics fluctuate markedly, DNN inference remains necessary. Therefore, the *when* aspect of adaptive inference calls for an estimation module that continuously monitors the volatility of the video analytics and, in turn, signals when inference can be skipped. In turn, quantifying the volatility allows for estimating the length of the skip window. This length denotes *how many* consecutive frames can be bypassed based on the measured volatility as an overly long window may overlook sudden events, whereas one that is too short limits the latency and energy savings.

The red line in Fig. 3 illustrates our "fake inference" approach. Specifically, in the bottom plot, it is evident that our method closely follows the trend of the default execution. A finer-grained analysis (Fig. 3 top plot) reveals that, although our method occasionally misses some values, it maintains strong overall performance. According to our analysis, this approach results in only a 5.1% precision reduction compared to the default execution. Furthermore, when grouping the results by second (a reasonable aggregation given the nature of the use-case) the imprecision difference between real and fake inference is merely 2.52% with a skipping ratio (network traffic reduction) of 74% of the total inference tasks.

3 The FakeInf Framework

3.1 Overview

The FakeInf framework has been designed to seamlessly attach to the pre- and post-processing stages of existing DL model serving pipelines. FakeInf introduces a lightweight decision module that monitors the stream-level analytics (e.g., object counts) produced by the DNN. Its goal is twofold: (i) decide whether the next frames truly require inference; and (ii) determine how long a skip window can safely last based on supplied confidence levels given by users that indicate their tolerance to imprecision.

Figure 4 provides a high-level overview of how FakeInf functions. In brief, a *warm-up* phase is first executed (Fig. 4 ①). During this phase every incoming frame passes through the FakeInf *FrameFilter* and is forwarded to the DNN. While these early results arrive, FakeInf compiles a statistical estimator that summarises recent analytics

to *learn* and *quantify* stream volatility. Once warm-up ends, FakeInf enters the *steady-state* phase (Fig. 4 ②), where the following algorithmic process is adopted. For each new frame FakeInf (i) computes a volatility score from the estimator; (ii) updates an internal "confidence" counter via a Reinforcement Learning (RL)-inspired rewarding mechanism; and (iii) selects an action: either "infer" or "skip for X frames". If volatility stays below a user-defined configurable imprecision threshold (introduced in subsection 3.2) and confidence is high, the FrameFilter suppresses DNN execution for the next X frames, where X grows with confidence. During a skip window, the system reuses the last DNN output. When volatility rises and the estimator's predictions deviate from the real scene, confidence drops and the FrameFilter immediately rolls back and resumes full DNN inference.

Hence, by avoiding unnecessary DNN passes, FakeInf reduces GPU load, energy draw, and the volume of downstream data. Operators can tune algorithm confidence threshold as well as the min and max skip length to balance accuracy against latency and cost, ensuring compliance with application-level QoS targets.

3.2 Algorithmic Process

Algorithm 1 provides an overview of the decision logic employed by FakeInf to quantitatively determine the skip length for adaptive DNN inference. Several estimators can be used to track the short-term dynamics of a data stream. Nevertheless, any candidate must impose minimal computational overhead as the adaptive logic is valuable only if its (computational) cost is far lower than executing a full DNN inference. Prominent examples of lightweight estimators are moving averages, such as the Exponential Weighted Moving Average (EWMA) that offers a balance between responsiveness and cost. The EWMA updates in O(1) time by requiring only the previous estimate and the current datapoint (e.g., video frame) rather than a full history of observations. Given the current datapoint v_i and previous estimate μ_{i-1}, the EWMA is updated as follows:

$$\mu_i = \alpha \cdot v_i + (1 - \alpha) \cdot \mu_{i-1},\ \alpha \in [0, 1] \tag{1}$$

The smoothing parameter α controls the estimator's memory. A larger α places more weight on recent frames, while a smaller α produces a smoother signal but may lag behind fast variations. Because the algorithm stores only the latest μ_i, it imposes negligible memory overhead, making it suitable for high-throughput video streams processed in situ on MEC nodes.

Note that as our goal is to capture the volatility of the emitted analytics, the absolute value of each metric (e.g., car count) is less informative than its frame-to-frame change. We therefore actually operate on the first-order difference $\delta_i = |v_i - v_{i-1}|$, where large values of δ signal a sudden shift in scene dynamics, whereas small values indicate stability. Moreover, while the EWMA is inexpensive, it responds poorly to outliers. A single spike can pull the estimate upward for many frames, and, conversely, the estimator may lag behind when a stable period abruptly ends. To address this limitation we adopt a Probabilistic EWMA (PEWMA) as shown below:

$$\begin{aligned} \tilde{\alpha}_i &= \alpha - P_i, \alpha \in [0, 1] \\ \mu_i &= \tilde{\alpha} \cdot \delta_i + (1 - \tilde{\alpha}) \cdot \mu_i \end{aligned} \tag{2}$$

For our PEWMA, the smoothing factor $\tilde{\alpha}_i$ adapts to how "typical" the current δ appears. Specifically, for each δ_i we estimate its likelihood P_i with a Gaussian kernel density estimator, avoiding any prior assumption about the underlying distribution. The effective weight then becomes $\tilde{\alpha}_i = \alpha - P_i$. With this, unusual deltas (low P_i) increase the influence of the current observation, allowing the estimate to track rapid changes; whereas common deltas (high P_i) reduce the weight, preserving stability [25]. Thus, the PEWMA retains the O(1) update cost of EWMA, requires no pre-defined window, while reacting quickly to both transient spikes and longer-term shifts.

With the PEWMA serving as the volatility tracker and used to provide an estimation for the distance of the next delta ($\hat{\delta}_{i+1} \leftarrow \mu_i$) each incoming frame updates the estimator in constant time. In turn, FakeInf keeps a rolling standard deviation σ_i of the deltas, maintained with an incremental (Welford-style) update that also runs in O(1) time and requires no historical buffer [27]. After these updates, the framework computes a *confidence* score:

$$c_i = 1.0 - \frac{|\hat{\sigma}_i - \sigma_i|}{\sigma_i},\ c_i \leq 1.0 \tag{3}$$

where $\hat{\sigma}_i$ is the standard deviation of the PEWMA estimator and σ_i is the observed rolling value. A high c_i indicates $\hat{\sigma}_i \rightarrow \sigma_i$ and the estimator is following the actual stream behavior, while a low c_i reflects a growing error. The confidence acts as the reward signal in FakeInf RL-based control loop. When successive frames yield high rewards, the algorithm accumulates credit and, once a threshold is crossed, issues a "skip X frames" action. If confidence drops, the credit decays immediately, forcing the FrameFilter to resume full DNN inference to preserve accuracy and satisfy QoS targets.

Capitalizing on the RL-inspired credit-decay loop, Equation 4 is used to compute the size of the skip S_{i+1} for the next frame. This is governed by the confidence score c_i, which captures how closely the estimator follows the current video stream dynamics. After computing c_i, FakeInf compares it with a user-given imprecision tolerance $\gamma \in [0, 1]$ and specifically the threshold $1 - \gamma$. Here, γ approximates the max prediction error the user can tolerate (e.g., $\gamma = 0.15$). This parameter sets the system's sensitivity when choosing the next skip window $S_{i+1} \in \mathbb{Z}^+$, bounded by $S_{\min}$ (typically 1, meaning no skip) and a configurable $S_{\max}$ set by frame-rate or latency constraints. Specifically, if $\gamma \rightarrow 0$, the framework behaves conservatively, only near-perfect confidence allows frame skipping, so the pipeline effectively runs with continuous inference (no skip). On the other hand, if $\gamma \rightarrow 1$, the framework becomes eager to save resources and will shorten or extend the skip window at every decision point, even when confidence is at a low point. Hence, whenever c_i falls below the user acceptable tolerance, FakeInf immediately reverts to continuous inference for the next interval to prevent accuracy loss. During the initial warm-up phase or after prolonged disconnections, FakeInf forces continuous inference until the confidence score stabilises to avoid oscillations.

$$S_{i+1} = \begin{cases} \min(S_{\max}, \lceil \frac{c_i}{1-\gamma} \rceil \cdot S_{\min}), & c_i \geq 1 - \gamma \\ S_{\min}, & \text{else} \end{cases} \tag{4}$$

In summary, by continuously adapting the skip length S_{i+1}, FakeInf reacts quickly to periods of high volatility while maximizing latency reduction and energy savings during stable scenes. The control loop executes in $O(1)$ time per frame as no history of the video stream is required beyond the statistical estimators. Moreover,

Algorithm 1 Adaptive Inference

Input: User imprecision tolerance γ and upd value v_i for current frame
Output: Skip length S_{i+1} and stream volatility estimates $\hat{\delta}_{i+1} \leftarrow \mu_i$ and c_i
Ensure: User requirements $\{S_{i+1} | S_{i+1} \in \mathbb{Z}^+ \text{ and } S_{i+1} \in [S_{min}, S_{max}]\}$

```
if t_i < T_warm then
    //compute distance and est error
    δ_i ← |v_i − v_{i−1}|
    ε_i ← δ_i − δ̂_i
    //update stream volatility estimations
    P_i ← ProbDistro(ε_i)
    δ̂_{i+1}, σ̂_{i+1} ← PEWMAandWelford(P_i, δ_i)
    //update RL-based confidence
    c_i ← calcConf(σ_i, σ̂_{i+1})
    if c_i ≥ 1 − γ then
        //compute skip length
        θ ← c_i/(1 − γ) · S_min
        if θ < S_max then
            S_{i+1} ← θ
        else
            S_{i+1} ← S_max
        end if
    else
        //low confidence, rollback to frame-to-frame inference
        S_{i+1} ← S_min
    end if
else
    //warm-up phase, no skipping only prepping volatility tracking
    S_{i+1} ← S_min
    δ_{i+1}, σ_{i+1} ← PEWMAandWelford(δ_i)
end if
return S_{i+1} //frames to skip
```

the imprecision-tolerance parameter γ is the only mandatory user-defined knob, although operators cal also tune the bounds $S_{\min}$ and $S_{\max}$ to suit app-specific latency or frame-rate constraints.

4 Use Case and FakeInf Implementation

4.1 Hardware Infrastructure

Our use-case is deployed on an infrastructure that combines a B5G network, edge computing, and high-resolution video sensing for real-time road traffic analysis. The network backbone is a commercial/private 5G network deployed by Orange Romania, configured for both Standalone (SA) and Non-Standalone (NSA) modes. The network can support multiple logical slices and in this case we selected an enhanced Mobile Broadband (eMBB) slice for high-throughput transmission of video streams from HD video cameras. Connectivity between the sensing equipment and the edge computing facility is established via a Nokia FastMile 5G Customer Premises Equipment (CPE), which handles traffic differentiation through multiple VLANs configured on a single SIM with multiple Data Network Name (DNN) profiles.

To process the video stream and emit real-time analytics, a GPU-enabled MEC node is deployed and configured within proximity of the road intersection. The MEC is equipped with an Intel Xeon Silver 4210 CPU @ 2.20GHz, 32GB of main memory, and a V100 NVIDIA GPU with 5,120 CUDA cores and 640 Tensor cores (peak FP16 performance 28 TFlops) and 32GB memory. Moreover, the edge server ingests and processes video data from multiple sources, and operates under strict latency constraints to provide actionable traffic insights. The architecture of the edge platform is designed according to the Service Enabler Architecture Layer (SEAL) framework from 3GPP Release 16, which ensures tight integration between network resources and edge applications. With this, the MEC is capable of handling concurrent HD video streams at a frame rate of at least 25 fps and maintaining a processing latency below 75ms.

The sensing layer consists of 8 Mobotix Mx-BC1A-4-IR-D bullet cameras strategically deployed within a crowded intersection in Iasi to capture comprehensive visual coverage of the traffic environment. These Power-over-Ethernet cameras provide high-quality resolution video with a horizontal field view over 90 degrees and an infrared (IR) range up to 30m, enabling effective operation in varying lighting conditions. The cameras are connected to an Aruba PoE switch, which aggregates the video streams and forwards them through the 5G CPE to the MEC node.

4.2 Use-Case Software Implementation

The high-level overview of the use-case software architecture follows the design depicted in Fig. 2. It relies on the open-source Savant framework[4], a Python abstraction layer built on top of the Nvidia DeepStream SDK for designing high-throughput, GPU-accelerated, and DL video processing pipelines for edge computing. Its architecture follows a directed acyclic graph (DAG) model, wherein video streams are captured via *source elements* supporting various protocols. In our case, these sources are the road-side IP cameras streaming over the B5G network. The incoming video streams undergo decoding via hardware-accelerated Nvidia's module (NVDEC), converting compressed inputs into raw frames for further analysis.

Moreover, the framework is able to seamlessly integrate DL inference engines, facilitating tasks such as object detection and classification. In our case, the input frames are forwarded to a YOLOv8-X detector (input 640 × 640 px) executed via TensorRT/ONNX. Additionally, video frames and their associated metadata can be processed through a sequence of user-defined pre- and post-inference filters, Python-based decoders or encoders, and metadata processors, enabling the application of sophisticated selection criteria based on object attributes, spatial or temporal constraints, dynamic application logic, and many more. We utilize this functionality in order to resize the incoming video streams, and keep useful data only the cars' bounding boxes and count. For communication between python modules (e.g., filters) and ML processing components, Savant employs an asynchronous messaging architecture based on ZeroMQ[5] with a PUB/SUB model, enabling scalable and decoupled data exchange within the pipeline. For integration with external systems, users must develop Python-based adapters to encode and disseminate results. In our case, we created an adapter that streams inference results to a cloud-based Kafka queue, enabling fast responses and seamless consumption by downstream services, including the cloud dashboard used by Iasi traffic operators.

The last component of our use case is the operator's dashboard (Fig. 5). Through the dashboard, operators can observe the video stream and the annotated frames, as well as statistics, like the number of cars in the road in real-time. Moreover, we integrated

[4] https://savant-ai.io/
[5] https://zeromq.org/

Figure 5: Use-Case Dashboard

the monitoring stack (see Sec. 4.4) with this dashboard for users to observe the underlying MEC node utilization metrics.

4.3 Integration with the FakeInf Framework

Processing video near the cameras on the MEC node reduces backhaul traffic and latency, yet the large YOLOv8-X model imposes substantial and highly variable GPU load when many cameras stream concurrently. Integrating FakeInf with Savant-based pipeline and throttling inference on-the-fly can keep latency and energy use within lower levels while preserving analytic accuracy.

To realize the FakeInf framework and its integration with the use case, we implemented two programming modules following the Python programming abstractions provided by the Savant framework. The first module operates as a pre-inference filter, whose functionality is to either discard or propagate frames to the subsequent stages of the pipeline based on the computed periodicity (see the *Selector* component of Fig. 4). In contrast, the core functionality of our framework is embedded in a post-inference filter. As described in Sec. 3, the FakeInf algorithm processes the relevant inference target metric (e.g., the car count) and updates its internal lightweight PEWMA *Statistical Estimator*.

For the *Warm-up Phase* of our framework (Fig. 4 ①), users provide a specific configuration defining the size of the datapoints that the system will use to train the corresponding *Statistical Estimator*. During this phase, the system exclusively performs ML inference on each incoming frame. Then, the system goes to the *Steady-state Phase* in which it computes the respective periodicity and disseminates it to *Selector*. To enable communication between our components, we employ a shared memory data structure. Through this structure, modules (both pre-filter and post-filter) have access to common data, where the pre-filter reads the current interval, while the post-filter writes the updated interval determined by the FakeInf analysis (Fig. 4 ②). To this end, the entire FakeInf code is implemented as a library for the Savant Framework, which users can enable with just a few lines of code.

4.4 Monitoring Stack

In order to have the required observability in our analysis, we desinged and implemented a comprehensive monitoring stack that captures both application-level metrics, like latency, and hardware metrics, like computational footprint and energy consumption.

Firstly, we focused on extracting app-level metrics using an internal mechanism of Savant framework. Specifically, the video frames in Savant have associated metadata that can be processed through Python-based metadata processors, enabling developers to apply programmatically new tags on them. We use this functionality

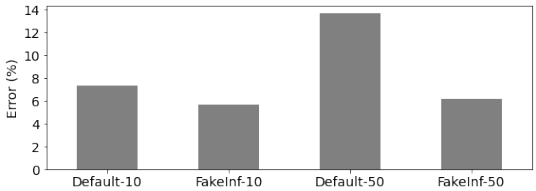

Figure 6: MAPE per Frame

to compute the inference latency for each frame by annotating the frame with a timestamp before entering the inference pipeline, and subtracting the timestamp after the inference.

In addition to monitoring app-level latency, we introduced a hardware monitoring stack to capture utilization metrics from the underlying execution environment, specifically from the Edge server. For this, we used open-source tools such as NetData and Prometheus[6] on the Edge server. These tools allow for the extraction of server utilization metrics (e.g., CPU/memory usage), service-level metrics (e.g., CPU usage per container), and network metrics (e.g., data transferred to the edge server). In addition, we created a wrapper on top of the nvidia-smi interface[7] to enable both GPU management (e.g., cap frequency) and GPU monitoring to extract critical metrics, such as power drawn, operating frequency, utilization and idle time, all specific to GPU of the MEC node.

5 Evaluation

To evaluate the efficacy and efficiency of FakeInf, we perform: (i) accuracy evaluation between the default (YOLOv8-X) deployment with different values of inference periodicity and FakeInf configurations; (ii) performance evaluation, comparing application latency, GPU usage, network traffic, and energy consumption, w/o FakeInf; (iii) model pruning vs FakeInf with a comparison against lighter models (YOLOv8-N and YOLOv8-M); and (iv) scalability study, evaluating FakeInf scalability against other techniques.

Each experiment is run for 1 hour, unless otherwise noted, and all experiments are connected to the IP camera video feed of the use-case featuring a 25 fps input data rate. Moreover, the warm-up period for all evaluation methods (incl. FakeInf) is set to 5min and upon request from the traffic operators, the maximum tolerable imprecision is set to 10% ($\gamma = 0.1$). We explicitly note that all ethical guidelines requested by the UC operators were adhered to. Specifically, experiment sessions were only conducted from the control room (restricted access), no personal data was monitored, and no video stream content was stored for later access.

5.1 Accuracy Evaluation

First, we evaluate the effect of FakeInf on the emitted analytics stream of the DL model. We consider the execution of the vanilla use-case (DNN without *FakeInf*) as the baseline (ground truth), and compare the following methods: (i) *Default-10*, executes inference every 10 frames (fixed-sampling) retaining the same output for intermediate frames; (ii) *Default-50*, similar but with fixed sampling of 50 frames; (iii) *FakeInf-10*, with a max skip window of 10 frames; and (iv) *FakeInf-50*, similar but with $S_{\max} = 50$. We denote that, for fairness, all methods are executed on the same video feed.

[6] https://www.netdata.cloud/ and https://prometheus.io/
[7] https://docs.nvidia.com/deploy/nvidia-smi/index.html

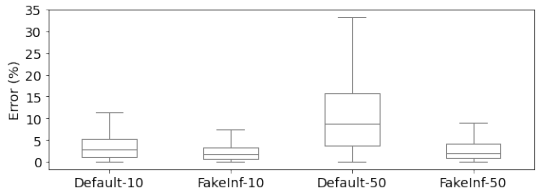

Figure 7: MAPE with Frames Aggregated per Second

Figure 6 shows the Mean Absolute Percentage Error (MAPE) of each method against the baseline. We observe the best results by FakeInf-10 with a MAPE of 5.13%, followed by FakeInf-50 with a MAPE of 6.19%. In contrast, the fixed sampling methods illustrate the worst results with Default-10 and Default-50 introducing a MAPE of 8.31% and 14.64%, respectively. Interestingly, even with $S_{\max} = 50$, FakeInf still achieves better precision than Default-10, as it exploits the low-volatility phases in the stream and *understands* that during high-volatility phases, adaptivity must roll back and execute inference at a per-frame instance.

To further investigate and eliminate (potential) synchronization factors between video frames, we also compute the mean car-count at a per second aggregation. Figure 7 depicts boxplots for the MAPE of the different methods. The results clearly show that FakeInf outperforms the fixed sampling methods. Specifically, FakeInf-10 achieves a median error of 1.74% and a mean error of 2.52%, while FakeInf-50 attains a median error of 2.05% and a mean error of 3.16%. In contrast, the Default-10 configuration exhibits a median error of 4.01% and a mean error of 5.56%. The poorest performance is observed for Default-50, with a median error of 9.96% and a mean error of 13.27%. We note that the aggregated results yield lower errors compared to the per-frame evaluation, as the aggregation process smooths out frame-level fluctuations and mitigates the influence of outliers on the overall car-count metric.

5.2 Resource Utilization Comparison

Next, we evaluate the runtime performance of the use-case w/o the integration of FakeInf. For these runs, we utilize our monitoring to capture inference latency, GPU usage, and energy consumption, as described in Sec. 4.4. In turn data volume reduction, which considers the network impact in kbps when disseminating the inference output from the MEC near the user equipment devices (IP cameras) to the use-case remote services (e.g., traffic dashboard).

Figure 8 presents four box plots comparing the baseline UC execution against the UC embedding FakeInf for the four performance indicators of interest. The first plot demonstrates that FakeInf reduces the mean latency by 59% compared to the baseline use case. In particular, the mean application latency is reduced from 53ms to 22ms. The second plot highlights a 66% reduction in GPU utilization, showcasing FakeInf's efficiency in minimizing computational demand. The third plot measures the energy consumption attributed to the edge server's GPU for DL model inference, where FakeInf achieves a 72% reduction. In terms of operating power levels this translates to a reduction from 61 W/s to 44 W/s. Finally, the fourth plot presents data volume reduction in terms of network traffic, revealing a 71% reduction, indicating significantly lower bandwidth usage (from 25.40 kbps to only 7.27 kbps). These results highlight FakeInf's ability to optimize system performance by reducing computational overhead, energy consumption, and network load while maintaining both responsiveness and accuracy of the DL models.

5.3 Model Pruning vs FakeInf Performance

In this set of experiments, we compare FakeInf against model compression via pruning. For a realistic comparison, we opt to run the use-case with FakeInf against the use-case adopting lighter model variants of YOLOv8, and specifically, the medium (25.9M params) and nano (3.2M params) variants. For fairness, *FakeInf*, *YOLOv8-M*, *YOLOv8-N*, and *YOLOv8-X* were all executed on the same video feed to measure differences in accuracy between the default execution and the under-comparison methods. We note that all plots compare the under-evaluation method (e.g., FakeInf) against the baseline (YOLOv8-X) and for visualization clarity, the boxplots that follow (Fig.11) present only the first 15 minutes of execution.

Figure 9 illustrates the mean per-frame error of each method, while Figure 10 presents boxplots of the errors aggregated over the car-count metric per second. FakeInf achieves a mean per-frame error of 6% and a mean per-second error of 2%. Interestingly, FakeInf significantly outperforms both YOLOv8-M and YOLOv8-N, which exhibit mean errors of approximately 13%/18% (grouped/frame-by-frame) and 33%/34% (grouped/frame-by-frame), respectively.

Nonetheless, one can advocate that although the lighter models do not achieve better accuracy than FakeInf, they may still exhibit a smaller computational footprint on the underlying MEC. To assess this assumption, we generate a series of boxplots (Fig. 11) illustrating the resource utilization of each approach. We should note here that we excluded network utilization because the results would be the same as the previous section (Sec. 5.2 and Fig. 8).

Interestingly, FakeInf not only demonstrates superior performance in terms of accuracy, but also imposes lower computational overhead and achieves lower application latency compared to the other methods. In particular, FakeInf outperforms the YOLOv8-N model (the lightest YOLO variant), achieving an average per-frame inference latency of 2ms compared to 4ms, alongside similar GPU utilization (5.4%-5.5%) and comparable power needs (41.5W for YOLOv8-N, 44.4W for FakeInf). The performance gap widens when comparing FakeInf to YOLOv8-M, with FakeInf delivering improvements of 56%, 30%, and 57% in latency, utilization, and energy consumption, respectively. To this end, FakeInf delivers a better balance of accuracy, latency, and resource efficiency, outperforming lighter YOLOv8 variants and demonstrating the effectiveness of targeted inference optimization over conventional model downsizing.

5.4 Scalability Evaluation

Next, we evaluate the scalability of the proposed use case when streaming multiple camera feeds over the 5G network and processing them on the MEC. These experiments examine: (i) if there are any bottlenecks (i.e., 5G network, MEC) in our infrastructure; and (ii) if FakeInf's algorithmic optimizations can enable efficiencies while increasing the number of camera feeds that must be processed. For the experiments, a script was coded so that the number of video streams fed to the MEC is increased by 1 every 10min until a max of 8 camera feeds is achieved. The input data rate received by the MEC is depicted in Figure 12. We denote that 8 camera

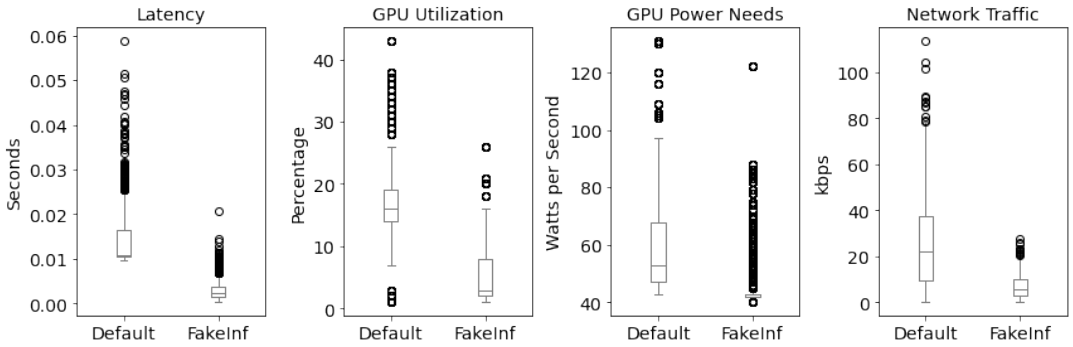

Figure 8: Utilization Metrics Comparison of the Default Use-Case vs the Use-Case with FakeInf Integration

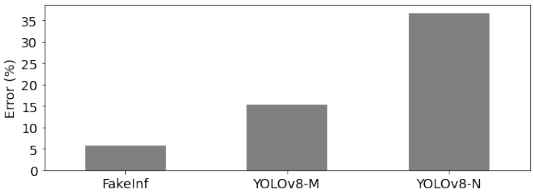

Figure 9: FakeInf vs Model Pruning Error Evaluation

Figure 10: FakeInf vs Model Pruning per Second Aggregation

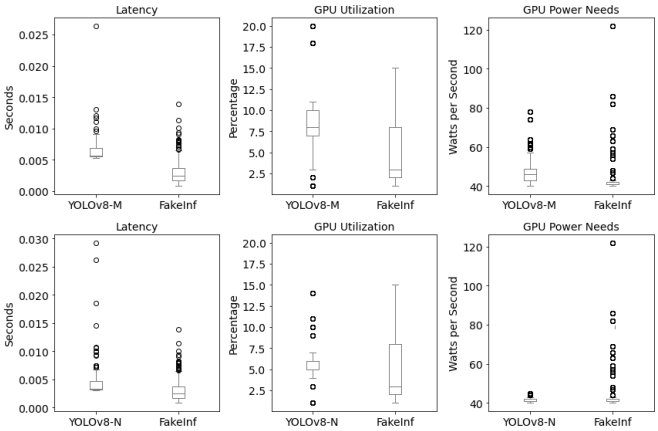

Figure 11: Utilization Metrics from Different Approaches

feeds is the max number available at the Iasi road intersection. In addition to the *Default* and *FakeInf*-enabled execution, we also evaluate a scenario where the use-case employs model swapping. For this we adopt the open-source ModelSwapper [24] where the DNN model variant is swapped-on-the-fly depending on the MEC resource availability and the desired latency set by the traffic operators. For consideration, the YOLOv8 N, M, and X variants were employed. Upon consultation with the use-case traffic operators, the max acceptable inference latency was set to 75ms. Beyond this,

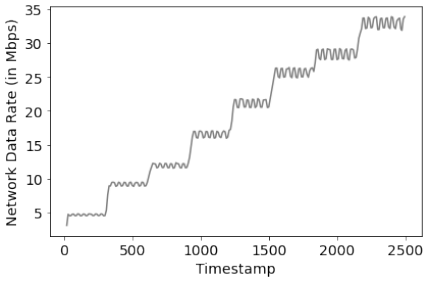

Figure 12: Multi-Camera Data Rate Received by MEC

the DL model serving pipeline is deemed unstable as the video feed received by the dashboard is "pixeled" and the inference error increases significantly.

Figures 13, 14, and 15 depict our key findings. The light gray line depicts results for the baseline use case setting, the dark gray line depicts the results for use-case with model swapping, and the black line depicts the results for the FakeInf optimization. Starting with Figure 13, the focus is given on latency where the max tolerable inference latency was set at 75ms (horizontal red dot line). With this QoS threshold in mind, one can immediately observe that the DL pipeline for the default use case reaches peak computational utilization (Figure 14), and experiences significant delays translating in negative QoS (Figure 13) after the introduction of the fourth video feed. In this case, the system becomes unstable, and the MEC can no longer handle the workload, rendering the results useless for traffic operators. In contrast, FakeInf and model swapping can support the requested 8 video camera feeds while remaining within the requested QoS threshold. Figures 13 and 14 show that the FakeInf optimization remains below 10ms for up to 5 video feeds and then gradually increases. However, it always stays below the threshold, as it effectively balances computational resource availability and inference accuracy. In turn, the Adaptive Model Selection approach can also support the 8 video feeds but presents significantly higher latency and computational footprint than FakeInf.

Moreover, Figure 15 depicts the power drawn by the GPU for the experiment series. From this, one can observe that the baseline starts with lower power consumption but gradually increases, reaching approx. 160W. The black line, corresponding to FakeInf optimization, maintains consistently lower power consumption compared to the other two approaches, staying mostly between 60W and 120W. Finally, the dark gray line, which represents model

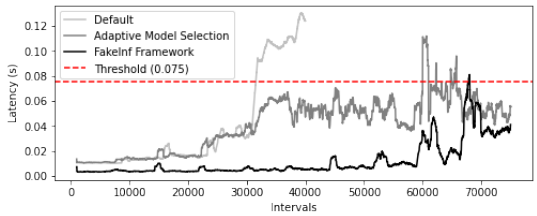

Figure 13: Latency per Frame of Scalability Evaluation

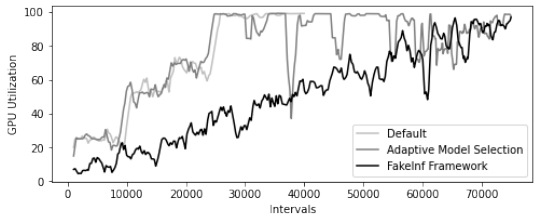

Figure 14: GPU Utilization of Scalability Evaluation

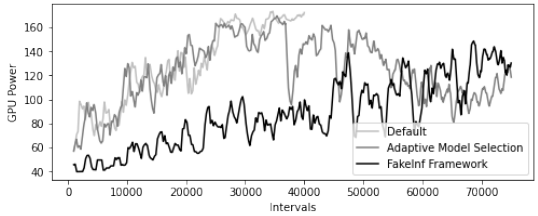

Figure 15: GPU Power of Scalability Evaluation

selection, follows a similar trend to the baseline but exhibits more fluctuations, including noticeable drops when the Nano variant is selected. We reiterate that execution with the Nano variant yields a 33% accuracy hit (section 5.3).

Finally, from the depicted results, it is evident that the 5G network provided by Orange Romania can satisfactorily support the 8 HD video feeds without introducing network delays. However, the MEC server struggles to support inference QoS requirements when scaling beyond 4 video feeds (no optimizations). This bottleneck is alleviated and the use-case can reach the 8 video feeds when FakeInf is integrated with the use-case.

Hence, our findings illustrate that *data-driven intelligent services like FakeInf can substantially enhance both network performance and sustainability, setting a strong foundation for future network generations (beyond 5G).* Moreover, despite the high data rates provided by current 5G infrastructures, *modern DL pipelines often fail to fully exploit them due to computational bottlenecks of edge computing nodes.* Smart support services such as FakeInf, can help overcome these limitations by enhancing the performance of EC devices and unlocking the full potential of 5G networks.

6 Related Work

Several techniques have been proposed to reduce latency for EdgeAI inference on DL model serving pipelines and, by extension, achieve energy savings. A broad category of techniques relies on model compression [10]. For example, *model pruning* algorithms compress the feature vector and execution time by removing less important or redundant parameters [26]. In turn, *model quantization* algorithms lower parameter precision (e.g., FP32 to INT8) to reduce the DNN memory footprint and computational overhead [17]. While model compression can succeed in reducing the inference delay, the gains come with a fixed cost and permanent accuracy loss even when only a modest relaxation would suffice. To overcome these limitations, *early exiting* can be introduced to DNNs. For example, Teerapittayanon et al. introduce BranchyNet [22] and Bateni et al. [4] ApNet, where auxiliary heads are added to the DNN for inference requests to exit the network if the classification is confident (BranchyNet) or when a latency budget is exceeded (ApNet). In contrast, FakeInf does not require modifications to the DNN and exploits efficiency savings when the algorithm is "confident" maintaining the original accuracy whenever volatility rises.

Another set of recent techniques suggest performing model selection at run time. For example, *distillation* transfers knowledge from a large "teacher" model to a compact "student" model that can be invoked when resources are scarce [14]. JellyFish [15] is a *hot swapping* technique where the algorithm selects a subset from a number of available DL models to run a batch of inference tasks to achieve high accuracy guarantees through the ensemble of these models. In turn, Trihinas et al. [24], introduce a low-cost classification algorithm that explores meaningful trade-offs between the computational overhead of the underlying edge node and energy consumption to select the model variant that will be used for a batch of inference tasks while maintaining user-desired classification accuracy. FakeInf differs from these techniques by working with a single model and thus, avoiding GPU memory pressure due to many resident variants; and its decision is per frame, driven by output dynamics rather than coarse resource metrics.

Recent video stream analysis techniques suggest employing a heuristic that skips the computation for the current frame when it closely resembles the previous one, allowing the DL model serving system to reuse previously inferred results. Ying et al. [30] suggest computing inter-frame similarity through motion vectors that capture pixel-level changes between consecutive frames. If the vector differences do not exceed a predefined threshold, their framework bypasses DL inference and reuses the previously cached output. Taranco et al. propose a similar scheme, denoted as δLTA [21]. However, their approach requires access to the surveillance cameras for the library to signal the downstream DL backend to lower the input data rate. Finally, Reducto [12] achieves low latency for video analytics by embracing offline training to determine discriminative features for a given set of queries and perform frame correlation filtering during inference. These methods rely on pixel-level similarity, which may miss semantic stability (e.g., the car count can remain steady while pixel differences fluctuate due to lighting) limiting skip windows and in turn, latency and energy savings.

7 Conclusion

In this paper, we introduced FakeInf, a lightweight and adaptive framework for optimizing the runtime inference of DL models on streaming data at the network edge. FakeInf intelligently monitors the volatility of analytic insights and adjusts the inference frequency to balance computational efficiency with application-level QoS. Through extensive experiments on a real-world smart traffic management use case, deployed on a 5G-enabled edge infrastructure, we demonstrated that FakeInf can substantially reduce application latency, network traffic, computational overhead, and energy consumption, while also maintaining high accuracy levels.

Furthermore, our scalability evaluation showed that FakeInf enables the processing of significantly more concurrent video streams without exceeding latency thresholds, thus overcoming the typical bottlenecks faced by MEC servers in such deployments. By empowering edge-based AI pipelines to operate more efficiently and sustainably, FakeInf contributes a practical step towards unlocking the full potential of next-generation intelligent networks, including beyond-5G and 6G architectures.

Our future directions are twofold. First, we intend to extend the algorithmic mechanisms of the decision-making process to extract and allow the use of seasonal data from the video stream so that traffic patterns can be used to increase the skip window with greater confidence. Second, we intend to explore the generalization of FakeInf to support a wider range of DL model types and application domains, as well as its integration with multi-tier cloud-edge orchestration strategies to further improve the resource efficiency of EdgeAI services.

Acknowledgments

This work is part of AdaptoFlow that has indirectly received funding from the European Union's Horizon Europe research and innovation action programme, via the TRIALSNET Open Call issued and executed under the TrialsNet project (Grant Agreement no. 101017141), respectively. In addition, certain language refinements were performed at the sentence level using ChatGPT. All original content and ideas are solely those of the authors.

References

[1] Jari Arkko, Michael Björn, Wolfgang John, Johan Sjöberg, Mattias Wildeman, Gustav Wikström, and Peter Öhlén. 2024. Beyond Bit-Pipes–New Opportunities on the 6G Platform. *Ericsson Technology Review* 2024, 6 (2024), 2–8.

[2] Lisa Asciak, Justicia Kyeremeh, Xichun Luo, Asimina Kazakidi, Patricia Connolly, Frederic Picard, Kevin O'Neill, Sotirios A Tsaftaris, Grant D Stewart, and Wenmiao Shu. 2025. Digital twin assisted surgery, concept, opportunities, and challenges. *npj Digital Medicine* 8, 1 (2025), 32.

[3] Akram Aslani and Mostafa Ghobaei-Arani. 2025. Machine learning inference serving models in serverless computing: a survey. *Computing* 107, 1 (2025), 47.

[4] Soroush Bateni and Cong Liu. 2018. ApNet: Approximation-Aware Real-Time Neural Network. In *2018 IEEE Real-Time Systems Symposium (RTSS)*. 67–79. doi:10.1109/RTSS.2018.00017

[5] Nicolae Cleju, Carlos Pascal, Ciprian-Romeo Comsa, Constantin-Florin Caruntu, Iulian B. Ciocoiu, Cristian Patachia-Sultanoiu, and Razvan Mihai. 2024. Towards Efficient Urban Mobility: Deployment Strategies for Smart Traffic Management and Crowd Monitoring Systems. In *2024 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. 997–1002. doi:10.1109/EuCNC/6GSummit60053.2024.10597063

[6] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 613–627.

[7] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernandez-Orallo. 2023. Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems* 38 (2023), 100857. doi:10.1016/j.suscom.2023.100857

[8] Tiago M. Fernandez-Carames and Paula Fraga-Lamas. 2024. Forging the Industrial Metaverse for Industry 5.0: Where Extended Reality, IIoT, Opportunistic Edge Computing, and Digital Twins Meet. *IEEE Access* 12 (2024), 95778–95819. doi:10.1109/ACCESS.2024.3422109

[9] Zhen Gao, Lei Yang, and Yu Dai. 2024. Large-Scale Cooperative Task Offloading and Resource Allocation in Heterogeneous MEC Systems via Multiagent Reinforcement Learning. *IEEE Internet of Things Journal* 11, 2 (2024), 2303–2321.

[10] Jangho Kim, Simyung Chang, and Nojun Kwak. 2021. PQK: Model Compression via Pruning, Quantization, and Knowledge Distillation. *CoRR* abs/2106.14681 (2021). arXiv:2106.14681 https://arxiv.org/abs/2106.14681

[11] Khaled B. Letaief, Wei Chen, Yuanming Shi, Jun Zhang, and Ying-Jun Angela Zhang. 2019. The Roadmap to 6G: AI Empowered Wireless Networks. *IEEE Communications Magazine* 57, 8 (2019), 84–90. doi:10.1109/MCOM.2019.1900271

[12] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. 2020. Reducto: On-Camera Filtering for Resource-Efficient Real-Time Video Analytics *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 359–376. doi:10.1145/3387514.3405874

[13] Nestor Maslej, Loredana Fattorini, Raymond Perrault, Yolanda Gil, Vanessa Parli, Njenga Kariuki, Emily Capstick, Anka Reuel, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Russell Wald, Tobi Walsh, Armin Hamrah, Lapo Santarlasci, Julia Betts Lotufo, Alexandra Rome, Andrew Shi, and Sukrut Oak. 2025. Artificial Intelligence Index Report 2025. arXiv:2504.07139 [cs.AI] https://arxiv.org/abs/2504.07139

[14] Yoshitomo Matsubara, Ruihan Yang, Marco Levorato, and Stephan Mandt. 2022. Supervised Compression for Resource-Constrained Edge Computing Systems. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2685–2695.

[15] Vinod Nigade, Pablo Bauszat, Henri Bal, and Lin Wang. 2022. Jellyfish: Timely Inference Serving for Dynamic Edge Networks. In *2022 IEEE Real-Time Systems Symposium (RTSS)*. 277–290. doi:10.1109/RTSS55097.2022.00032

[16] David A. Patterson, Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. 2021. Carbon Emissions and Large Neural Network Training. *CoRR* abs/2104.10350 (2021).

[17] Nadav Rotem, Jordan Fix, Saleem Abdulrasool, Garret Catron, Summer Deng, Roman Dzhabarov, Nick Gibson, James Hegeman, Meghan Lele, Roman Levenstein, Jack Montgomery, Bert Maher, Satish Nadathur, Jakob Olesen, Jongsoo Park, Artem Rakhov, Misha Smelyanskiy, and Man Wang. 2019. Glow: Graph Lowering Compiler Techniques for Neural Networks. arXiv:1805.00907 [cs.PL]

[18] Veronika Samborska. 2025. Scaling up: how increasing inputs has made artificial intelligence more capable. *Our World in Data* (2025). https://ourworldindata.org/scaling-up-ai.

[19] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: a GPU cluster engine for accelerating DNN-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 322–337. doi:10.1145/3341301.3359658

[20] Moysis Symeonides, Demetris Trihinas, George Pallis, Marios D Dikaiakos, Constantinos Psomas, and Ioannis Krikidis. 2022. 5g-slicer: An emulator for mobile iot applications deployed over 5g network slices. In *2022 IEEE/ACM Seventh International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 115–127.

[21] Raúl Taranco, José-María Arnau, and Antonio González. 2023. δ LTA: Decoupling Camera Sampling from Processing to Avoid Redundant Computations in the Vision Pipeline. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (Toronto, ON, Canada) *(MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 1029–1043. doi:10.1145/3613424.3614261

[22] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. 2017. BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks. *CoRR* abs/1709.01686 (2017). arXiv:1709.01686 http://arxiv.org/abs/1709.01686

[23] Demetris Trihinas, Panagiotis Michael, and Moysis Symeonides. 2024. Evaluating DL Model Scaling Trade-Offs During Inference via an Empirical Benchmark Analysis. *Future Internet* 16, 12 (2024), 468.

[24] Demetris Trihinas, Panagiotis Michael, and Moysis Symeonides. 2024. Towards Low-Cost and Energy-Aware Inference for EdgeAI Services via Model Swapping. In *2024 IEEE International Conference on Cloud Engineering (IC2E)*. 168–177. doi:10.1109/IC2E61754.2024.00026

[25] Demetris Trihinas, George Pallis, and Marios D Dikaiakos. 2018. Low-cost adaptive monitoring techniques for the internet of things. *IEEE Transactions on Services Computing* 14, 2 (2018), 487–501.

[26] Hanrui Wang, Zhekai Zhang, and Song Han. 2020. SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning. *CoRR* abs/2012.09852 (2020). arXiv:2012.09852 https://arxiv.org/abs/2012.09852

[27] B. P. Welford. 1962. Note on a Method for Calculating Corrected Sums of Squares and Products. *Technometrics* 4, 3 (1962), 419–420.

[28] Minrui Xu, Hongyang Du, Dusit Niyato, Jiawen Kang, Zehui Xiong, Shiwen Mao, Zhu Han, Abbas Jamalipour, Dong In Kim, Xuemin Shen, et al. 2024. Unleashing the power of edge-cloud generative AI in mobile networks: A survey of AIGC services. *IEEE Communications Surveys & Tutorials* 26, 2 (2024), 1127–1170.

[29] Vinay Yadhav, Andrew Williams, Ondrej Smid, Jimmy Kjällman, Raihan Ul Islam, Joacim Halén, and Wolfgang John. 2024. Benefits of Dynamic Computational Offloading for Mobile Devices.. In *CLOSER*. 265–276.

[30] Ziyu Ying, Shulin Zhao, Haibo Zhang, Cyan Subhra Mishra, Sandeepa Bhuyan, Mahmut T. Kandemir, Anand Sivasubramaniam, and Chita R. Das. 2022. Exploiting Frame Similarity for Efficient Inference on Edge Devices. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. 1073–1084.

[31] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proc. IEEE* 107, 8 (2019), 1738–1762.